THE C++ STANDARDIZATION COMMITTEE

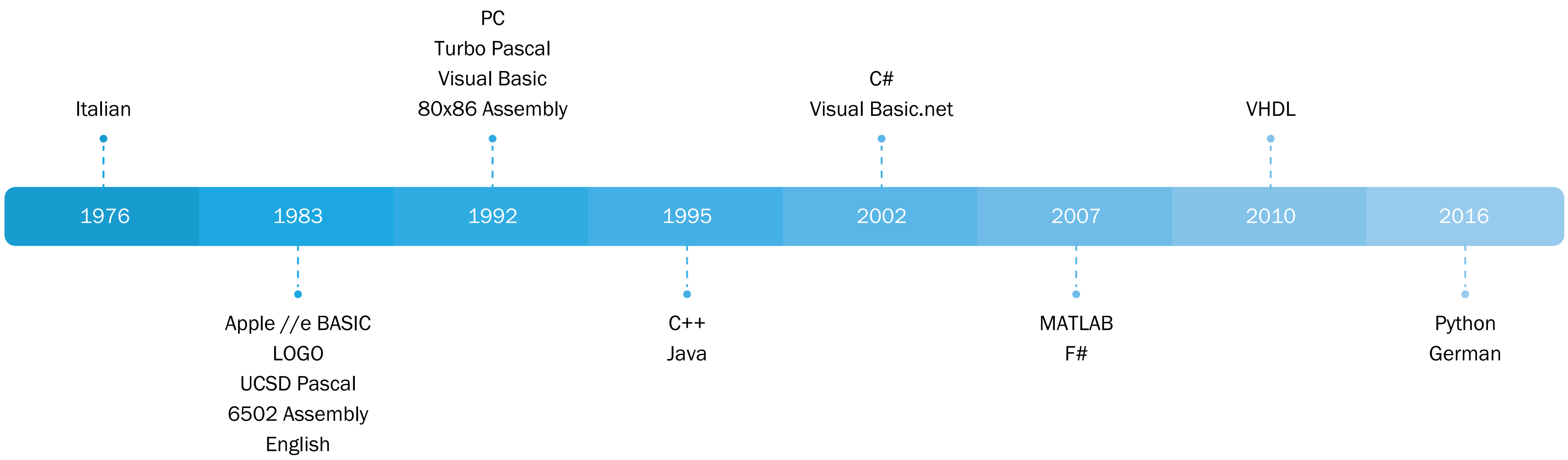
TALES FROM THE ITALIAN NATIONAL BODY

WHO ARE YOU?

- Marco Foco
- Born in Italy in 1975
- Computer Science Engineer
- Moved to Switzerland in 2015

- Native speaker of C++
- Occasional speaker of Italian, English

OH, SO YOU KNOW A BUNCH OF LANGUAGES



WHAT DO YOU DO?

- Scuba diving, repairing old computers, and studying for private pilot license
- Blogger
  - C++ explained to my dog (https://marcofoco.com)
  - C++ spiegato al mio cane (https://marcofoco.it)
- Streamer
  - https://twitch.tv/panspinningkids
  - https://pan.spinningkids.org
- Engineering Manager - NVIDIA
- Head of Italian Delegation for ISO/IEC JTC1/SC22/WG21
  - No, the cat didn't walk on my keyboard, it's the real name of the C++ standardization committee.
  - It's not even my first time: in 1997 I was among the founders of JIA – the Java Italian Association.

WHAT IS ISO/IEC JTC1/SC22/WG21???

- ISO
  - International Standard Organization
- IEC
  - International Electrotechnical Commission
- JTC1
  - Joint Technical Committee #1: Information Technology
- SC22
  - Sub-comittee #22: Programming languages, their environments and system software interfaces
- WG21
  - Working Group #21: C++

HOW DO YOU GET TO BE THE HEAD OF THE ITALIAN DELEGATION?

- You just create the delegation.
- It’s that simple.

OTHER ACTIVE WORKING GROUPS IN ISO/IEC JTC1/SC22

- WG5 – Fortran
- WG14 – C
- WG23 – Programming Language Vulnerabilities

IT'S MADE OF PEOPLE (REAL PEOPLE, I SWEAR!)

First meeting (1990)

C++11 Approval

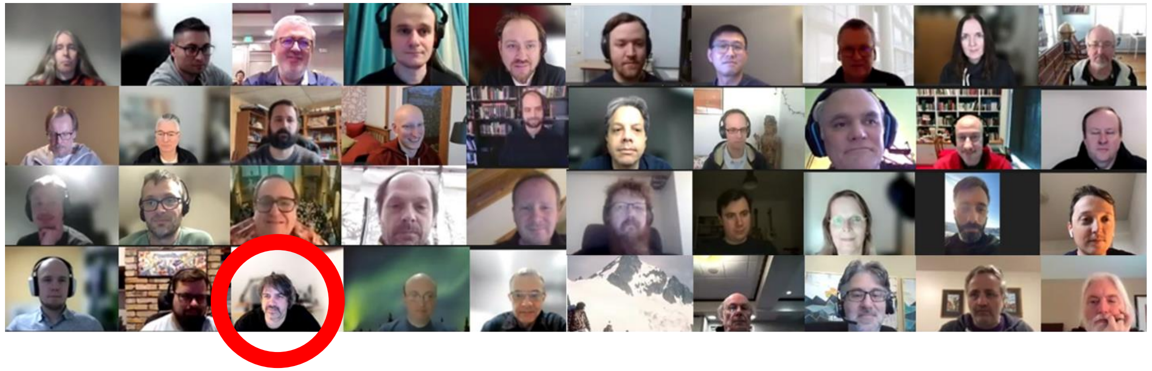
C++14, C++17

C++14 Approval

C++17 Approval

C++20 (JUST BEFORE THE PANDEMIC)

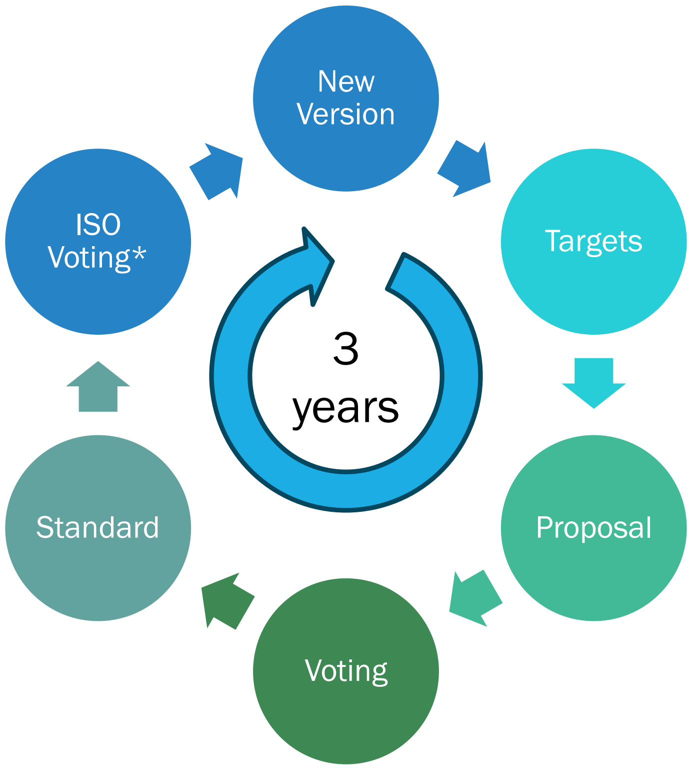
C++23 (HYBRID)



WHAT DOES THE ISO C++ COMMITTEE DO? AND WHEN? AND HOW?

- Tasks
  - Develops new C++ versions
  - Receives/creates and process Issues
- Meetings
  - Originally, in person, 3 times a year (sometimes using recurrent teleconferences, e.g monthly)
  - During pandemic: online, 3 times a year, and using teleconferences
  - Post-pandemic: hybrid, 3 times a year, and using teleconferences
  - Meetings last 1 week (Monday-Saturday)

THE (MODERN) C++ DEVELOPMENT CYCLE



COMPOSITION – MAIN GROUPS

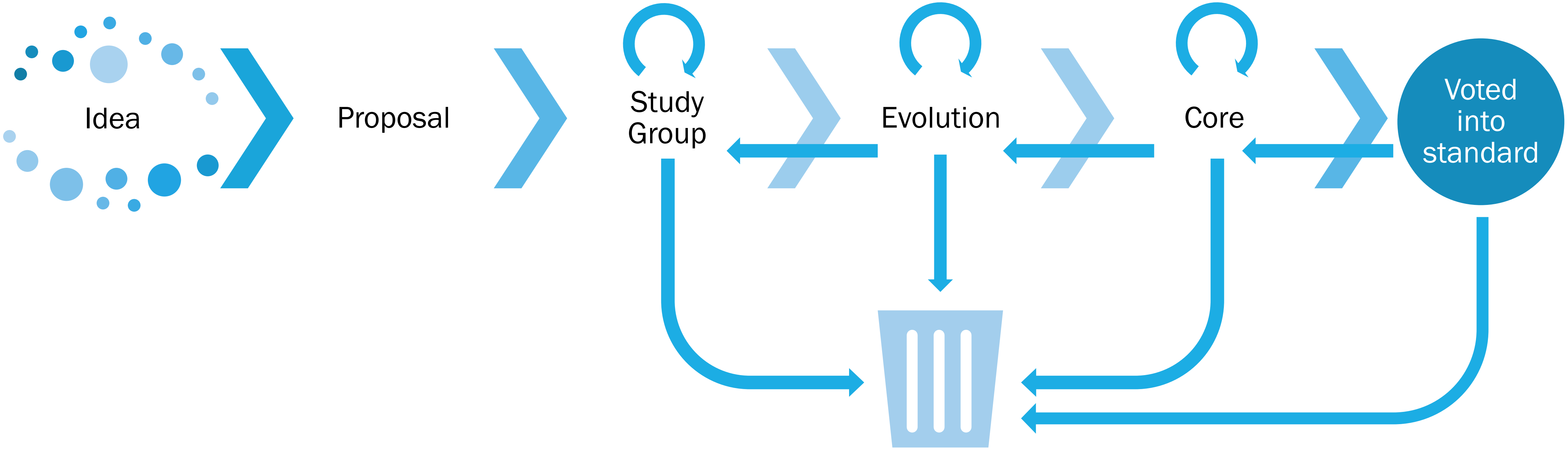
- AG – Admin Group
- DG – Direction Group
- ARG – ABI Review Group

- Language
  - EWG – Evolution Working Group
  - CWG – Core Working Group
- Library
  - LEWG – Library Evolution Working Group
  - LWG – Library Working Group

COMPOSITION – ACTIVE STUDY GROUPS (SG)

- SG1 – Concurrency
- SG4 – Networking
- SG6 – Numerics
- SG7 – Compile-time programming
- SG9 – Ranges
- SG10 – Feature Test
- SG14 – Game Development & Low Latency
- SG15 – Tooling
- SG16 – Unicode
- SG19 – Machine Learning
- SG20 – Education
- SG21 – Contracts
- SG22 – WG21/WG14 C/C++ Liaison
- SG23 – Safety & Security

EVOLUTION OF A PROPOSAL (PXXXXRY)

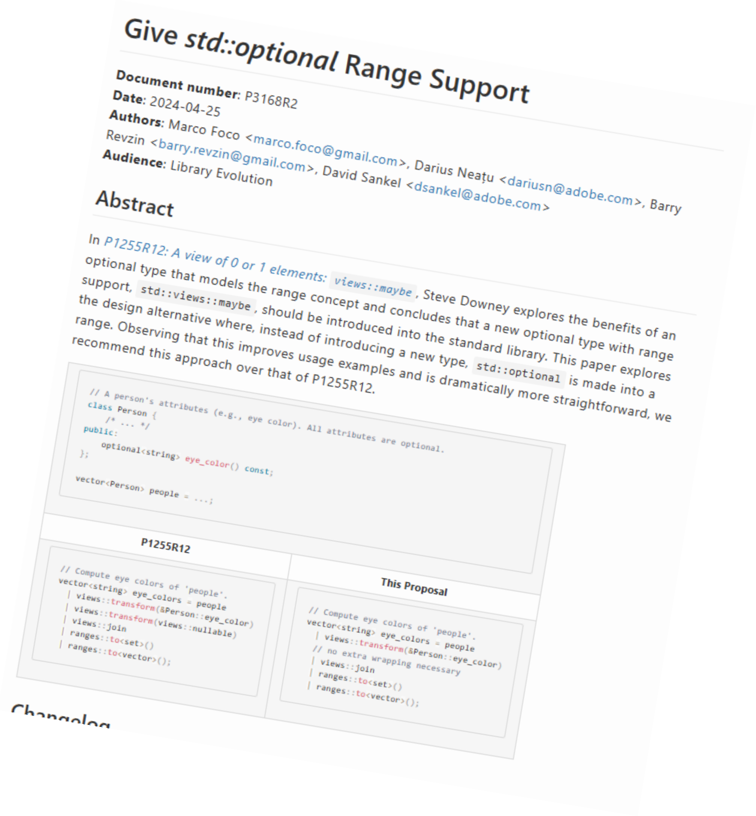


HOW LONG DOES IT TAKE FOR EACH PROPOSAL?

- At least 1 meeting per step
  - Study group
  - Evolution group
  - Core group
  - ➢ 1 year minimum
- Every other iteration will take longer, depends on:
  - The importance of the proposal
  - The load of work on the specific working group (in cases of overload on the Evolution groups, the “Incubators” are also enabled)
- For any complex proposal starting in one cycle, **the usual target is the end of next cycle**, or the one after
  - So every proposal takes on at least 4-8 years to get into the standard

CAN YOU SPEEDRUN IT?

- Yes.
- Case study – P3168: Give *std::optional* Range Support
  - Written in Febryary 2024 (P3168R0)
  - Was a derived work of P1255R12, so it could skip the SG and went directly to LEWG
  - Was *extremely* simple and concise, so was approved right away (Tokyo, March 2024)
  - Wording was iterated through conference calls in April (P3168R1 and R2)
  - Next meeting, it went to LWG, and was voted into the standard (St.Louis, June 2024)
- ➢ In 4 months the proposal went in the standard



Give *std::optional* Range Support

Document number: P3168R2
Date: 2024-04-25
Authors: Marco Foco <marco.foco@gmail.com>, Darius Neațu <dariusn@adobe.com>, Barry Revzin <barry.revzin@gmail.com>, David Sankel <dsankel@adobe.com>
Audience: Library Evolution

Abstract

In *P1255R12: A view of 0 or 1 elements:* `views::maybe`, Steve Downey explores the benefits of an optional type that models the range concept and concludes that a new optional type with range support, `std::views::maybe`, should be introduced into the standard library. This paper explores the design alternative where, instead of introducing a new type, `std::optional` is made into a range. Observing that this improves usage examples and is dramatically more straightforward, we recommend this approach over that of P1255R12.

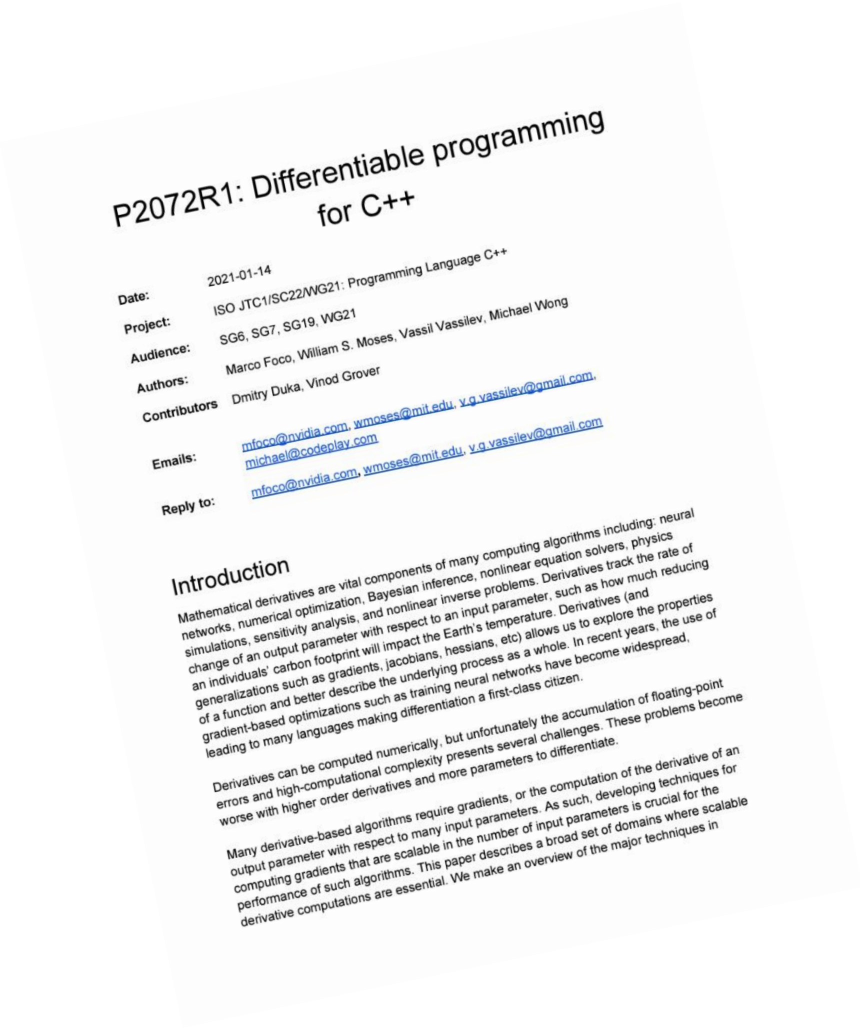
```
// A person's attributes (e.g., eye color). All attributes are optional.
class Person {
    /* ... */
public:
    optional<string> eye_color() const;
};

vector<Person> people = ...;
```

P1255R12	This Proposal

`// Compute eye colors of 'people'.`
`vector<string> eye_colors = people`
`  | views::transform(&Person::eye_color)`
`  | views::transform(views::nullable)`
`  | views::join`
`  | ranges::to<set>()`
`  | ranges::to<vector>();`
	`// Compute eye colors of 'people'.`
	`vector<string> eye_colors = people`
	`  | views::transform(&Person::eye_color)`
	`  // no extra wrapping necessary`
	`  | views::join`
	`  | ranges::to<set>()`
	`  | ranges::to<vector>();`

ANOTHER EXAMPLE...

- Written in 2020 (just before the pandemic, good timing!)
- Proposed introducing a new feature (differentiability of functions)
  - Complex
  - Difficult to understand
  - Difficult to implement
  - Very specific for a subset of users
  - Not well received
- Re-written in 2021, with new development in the sector
  - Still not well received
- Investigated for another couple of years
- Went into the trash bin



P2072R1: Differentiable programming for C++

Date: 2021-01-14
Project: ISO JTC1/SC22/WG21: Programming Language C++
Audience: SG6, SG7, SG19, WG21
Authors: Marco Foco, William S. Moses, Vassil Vassilev, Michael Wong
Contributors Dmitry Duka, Vinod Grover

Emails: mfoco@nvidia.com, wmoses@mit.edu, v.g.vassilev@gmail.com, michael@codeplay.com

Reply to: mfoco@nvidia.com, wmoses@mit.edu, v.g.vassilev@gmail.com

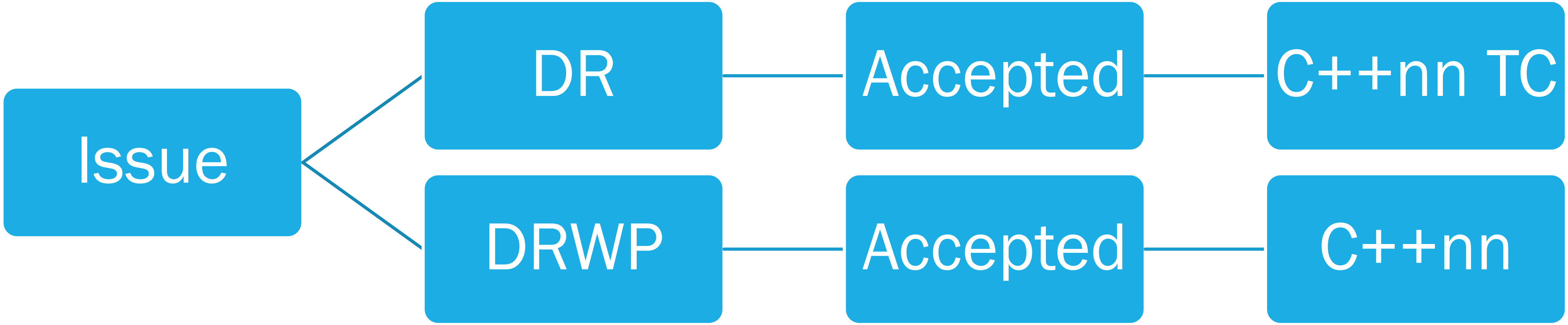
Introduction

Mathematical derivatives are vital components of many computing algorithms including: neural networks, numerical optimization, Bayesian inference, nonlinear equation solvers, physics simulations, sensitivity analysis, and nonlinear inverse problems. Derivatives track the rate of change of an output parameter with respect to an input parameter, such as how much reducing an individuals' carbon footprint will impact the Earth's temperature. Derivatives (and generalizations such as gradients, jacobians, hessians, etc) allows us to explore the properties of a function and better describe the underlying process as a whole. In recent years, the use of gradient-based optimizations such as training neural networks have become widespread, leading to many languages making differentiation a first-class citizen.

Derivatives can be computed numerically, but unfortunately the accumulation of floating-point errors and high-computational complexity presents several challenges. These problems become worse with higher order derivatives and more parameters to differentiate.

Many derivative-based algorithms require gradients, or the computation of the derivative of an output parameter with respect to many input parameters. As such, developing techniques for computing gradients that are scalable in the number of input parameters is crucial for the performance of such algorithms. This paper describes a broad set of domains where scalable derivative computations are essential. We make an overview of the major techniques in

WHAT IS A DEFECT REPORT



EXAMPLE OF DEFECT REPORT

2892. Unclear usual arithmetic conversions

Section: 7.4 [expr.arith.conv] **Status:** DR **Submitter:** Xxxxx Xxxxxx Date: 2024-05-06

[Accepted as a DR at the June, 2024 meeting.]

Subclause 7.4 [expr.arith.conv] bullet 1.4.1 specifies:

- If both operands have the same type, no further conversion is needed.

What does "needed" mean?

Proposed resolution (approved by CWG 2024-05-31):

Change in 7.4 [expr.arith.conv] bullet 1.4.1 as follows:

- If both operands have the same type, no further conversion is ~~needed~~ performed

WHAT DOES A NATIONAL BODY DO?

- Represent the interest of a country in the standardization
- Organize events
- Discuss proposals
- Submit Defect Reports and comments to the WP
- Approve standard documents (when they’re ready…)

HOW WE DO IT

- Regular monthly meetings
  - Normal discussion of organization
  - Discuss ideas for proposals
- Pre-plenary meetings during the meetings (Friday Evening)
  - Discuss controversial items
- Mailing list
  - Asynchronous discussion on ideas and proposals

EXAMPLE: HOW A PROPOSAL IS BORN

- Our experience
- Large, relatively old codebase
- Many C-isms

THE PROBLEM

- Use of strlen

```
void myFunction(const char* filename) {
 // […]
 auto x = strlen(filename);
 // […]
}
```

FIRST SOLUTION

- Let’s use strnlen_s, the greatest and safest version of strlen

- Requires an upper boundary of the possible length, requires to change all the APIs

```
void myFunction(const char* filename, int max_size) {
// […]
    auto x = strnlen_s(filename, max_size);
// […]
}
```

- But wait… we’re working with C++!

SECOND SOLUTION

- Change those functions to use std::string_view

```
void myFunction(std::string_view filename) {
 // […]
 auto x = filename.length();
 // […]
}
```

- This also has the extra issue of requiring to add null termination (in some cases), but we won’t discuss this today.

NEW PROBLEMS

- What happens now?

```
void myFunction(std::string_view filename);

void myOtherFunction(const char* filename) {
 myFunction(filename);
}
```

BASIC_STRING_VIEW'S CONSTRUCTORS

- `constexpr basic_string_view(const CharT* s);`

  "Constructs a view of the null-terminated character string pointed to by s, not including the terminating null character. The length of the view is determined as if by Traits::length(s). The behavior is undefined if [s, s + Traits::length(s)) is not a valid range. After construction, data() is equal to s, and size() is equal to Traits::length(s)."

- Which is equivalent to calling `strlen`. Not even `strnlen_s`.

C++26 EFFORTS TO MAKE C++ SAFER

- P2687 – Design Alternatives for Type-and-Resource Safe C++Profiles (B. Stroustrup, G. Dos Reis)
- P2816 – Safety Profiles: Type-and-resource Safe Programming in ISO Standard C++ (B. Stroustrup, G. Dos Reis)
- P3038 – Concrete suggestions for initial Profiles (B. Stroustrup)
- P3274 – A framework for Profiles development

THERE’S A MEMORY SAFE PROFILE FOR THAT!

- Ah! That’s the solution!
- Change the behavior of `string_view` in the **new memory profile** to NOT build from a `const char *`!!!
- Keep the behavior intact in the old (unsafe) standard profile.

MORE PROBLEMS

- Was myFunction(“test.txt”) ever problematic?
- “test.txt” type is char[9] – a type that embeds the maximum size of it
- We don’t need to remove all the cases where `const char*`, constructor is used just those when it’s really a **`const char*`**

HOW?

- Example of constructor that catches the char[N] case

```
template<typename CharT, typename Traits>
class basic_string_view {
 template<size_t N>
 basic_string_view(char (&s)[N]) {
     // check the length of a string in a bounded to N
 }
}
```

// CHECK THE LENGTH OF A STRING IN A BOUNDED TO N

- Did you notice, in the standard library we had the equivalent of `strlen`, but not of `strnlen_s`?

- Meet `char_traits::length_s`!

- It's the equivalent of `char_traits::length`, but it only works for types that has an embedded size (`char[N]`, `span<T, N>`, ...).

- It has a 2-parameter overload that is able to calculate the length of a `CharT*` in a bounded way (the limit is specified in the second parameter)

```
constexpr size_t length_s(const CharT*, size_t limit) { ... }
```

CONCLUSION

- This proposal will make `std::string_view` *conditionally* safer (under the *memory safe* profile)

- All the discussion applies to `std::string` as well

QUESTIONS?

- Questions?

- Not all at once...